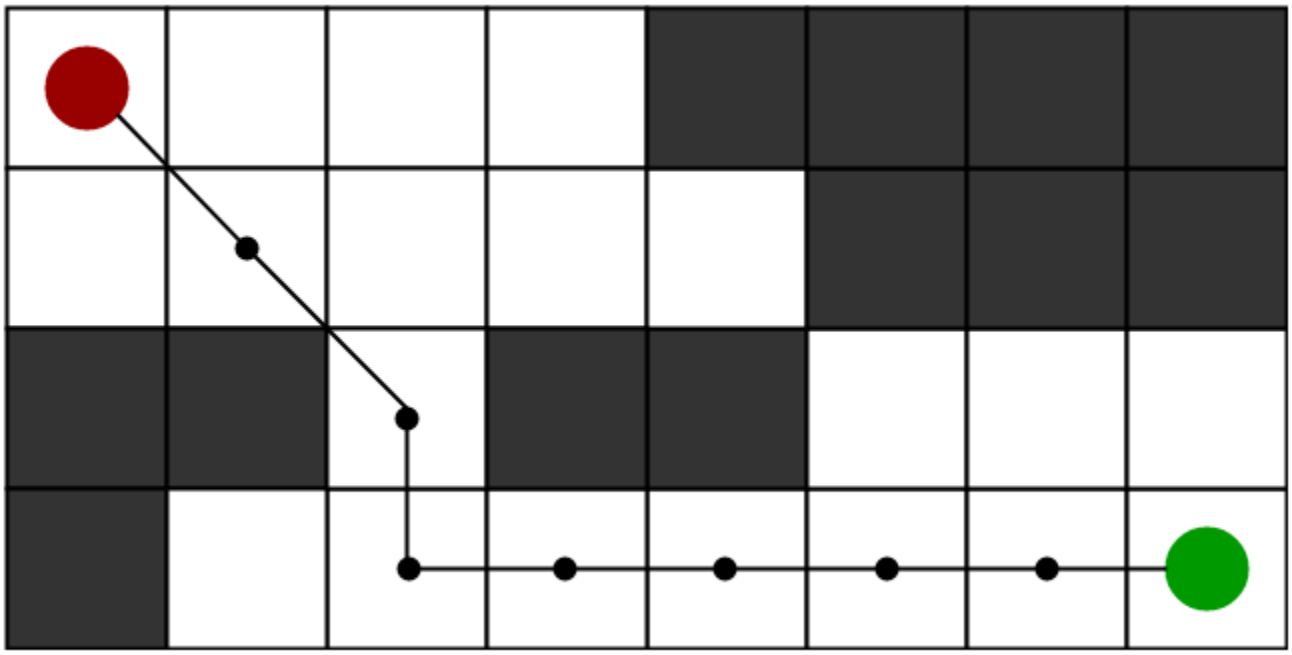
A* Search Algorithm

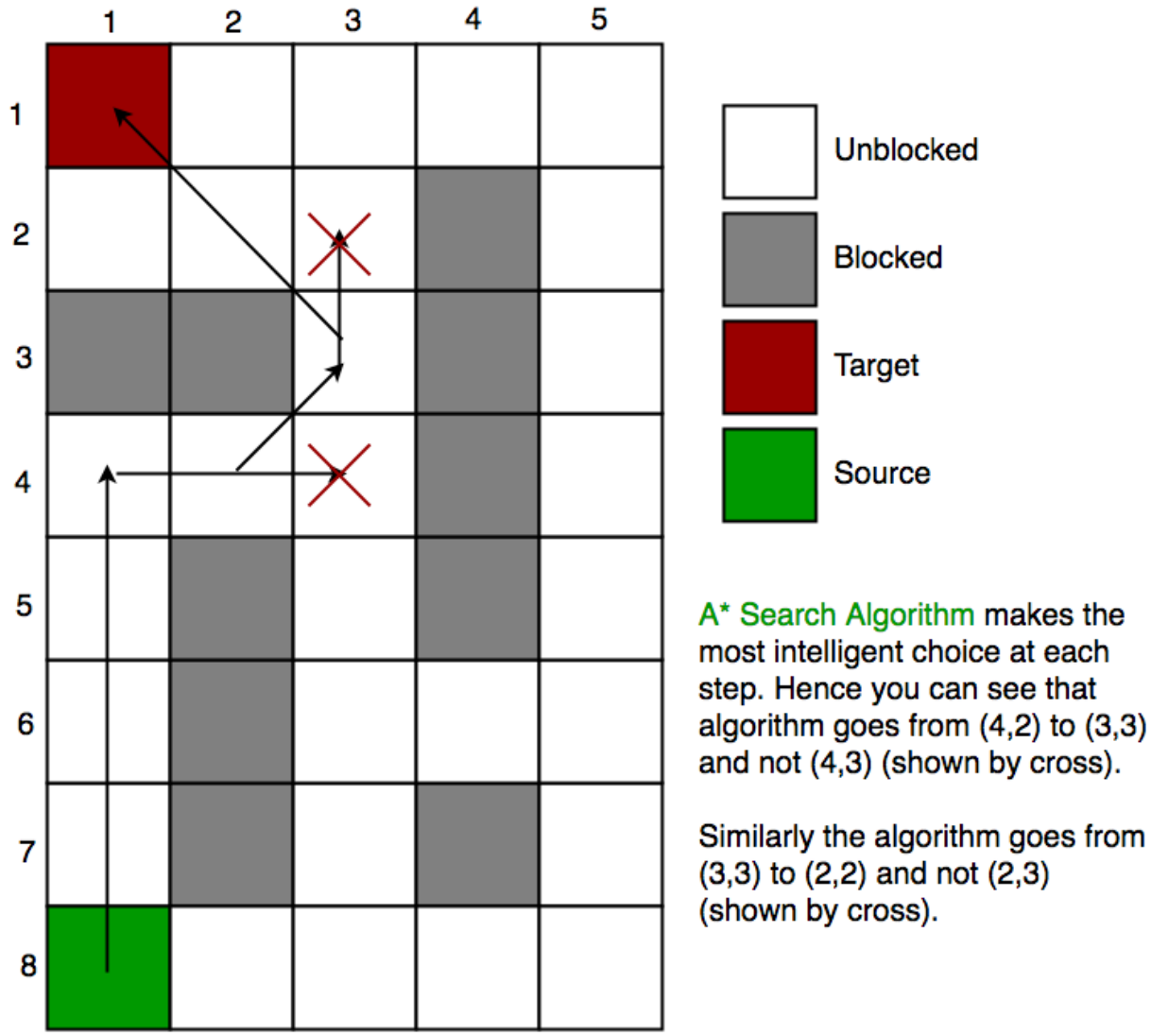
Motivation
To approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances.
We can consider a 2D Grid having several obstacles and we start from a source cell (colored red below) to reach towards a goal cell (colored green below)



What is A* Search Algorithm?
A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.
Why A* Search Algorithm?
Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms.
And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).

Explanation
Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue. What A* Search Algorithm does is that at each step it picks the node according to a value-'**f**' which is a parameter equal to the sum of two other parameters – '**g**' and '**h**'. At each step it picks the node/cell having the lowest '**f**', and process that node/cell.
We define '**g**' and '**h**' as simply as possible below
g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.
h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don't know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this 'h' which are discussed in the later sections.

So, suppose as in the below figure if we want to reach the target cell from the source cell, then the A* Search algorithm would follow path as shown below. Note that the below figure is made by considering Euclidean Distance as a heuristic.



Heuristics

We can calculate **g** but how to calculate h ?

We can do things.

A) Either calculate the exact value of h (which is certainly time consuming).

OR

B) Approximate the value of h using some heuristics (less time consuming).

We will discuss both of the methods.

A) **Exact Heuristics** –

We can find exact values of h, but that is generally very time consuming.

Below are some of the methods to calculate the exact value of h.

1) Pre-compute the distance between each pair of cells before running the A* Search Algorithm.

2) If there are no blocked cells/obstacles then we can just find the exact value of h without any pre-computation using the distance formula/Euclidean Distance

B) Approximation Heuristics –

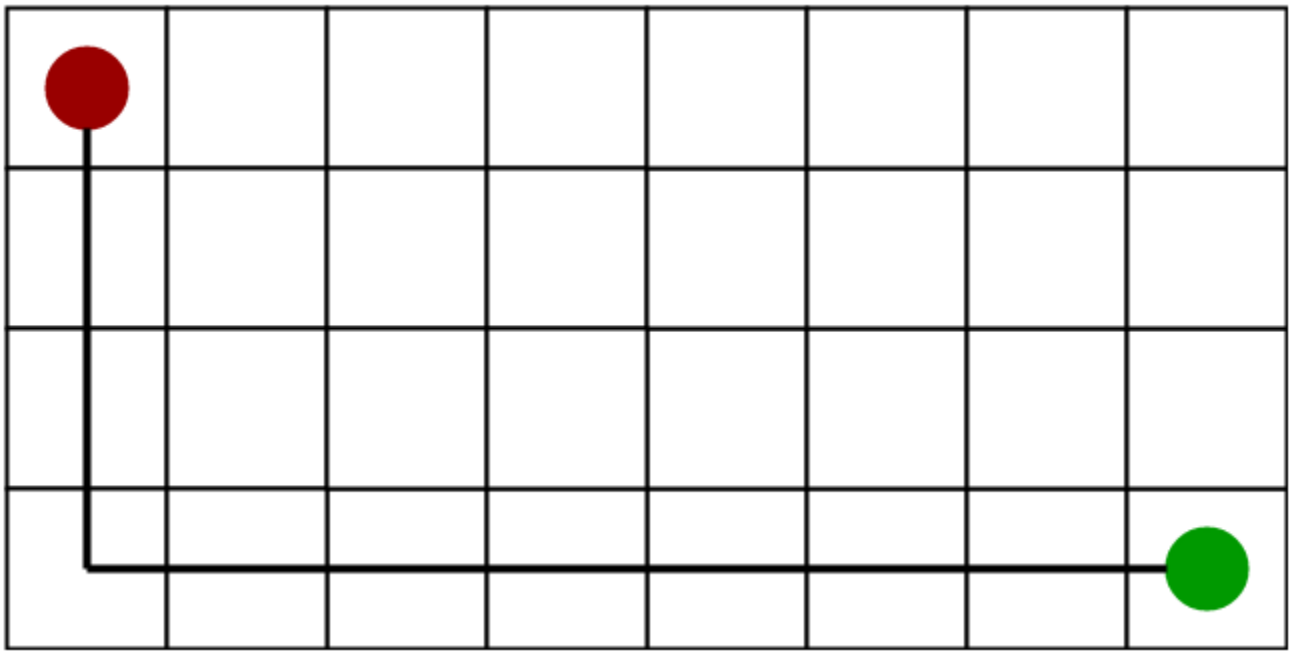
There are generally two approximation heuristics to calculate h –

1) Manhattan Distance –

- It is nothing but the sum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,

```
h = abs (current_cell.x – goal.x) +
    abs (current_cell.y – goal.y)
```

When to use this heuristic? – When we are allowed to move only in four directions only (right, left, top, bottom)



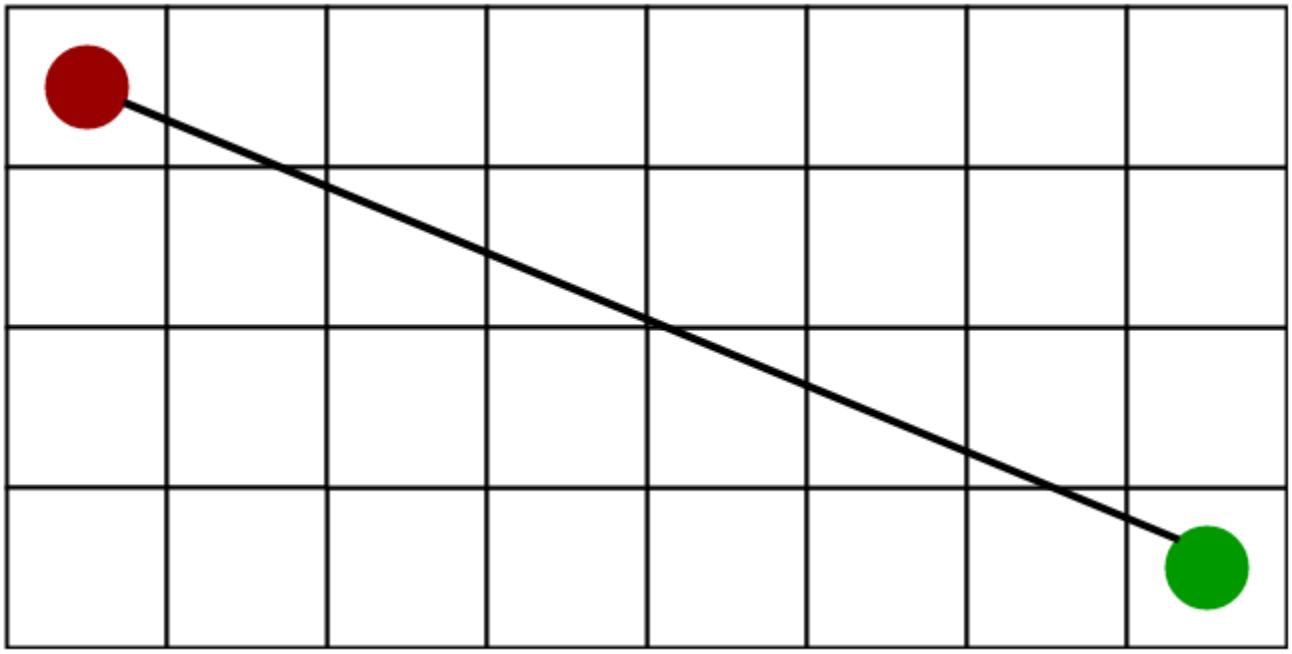
2) Euclidean Distance-

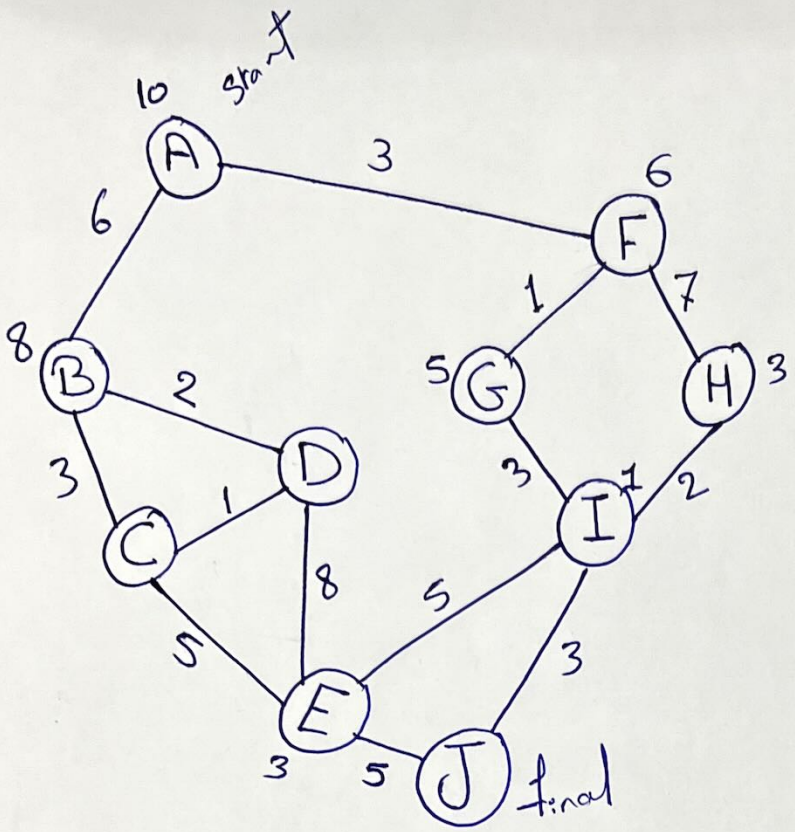
- As it is clear from its name, it is nothing but the distance between the current cell and the goal cell using the distance formula

```
h = sqrt ((current_cell.x – goal.x)2 +
          (current_cell.y – goal.y)2 )
```

- When to use this heuristic? – When we are allowed to move in any directions.

The Euclidean Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).





Step:1 :- Start at node A :-

Node B & F can be reached from A

A* algorithm calculates :- $f(n) = g(n) + h(n)$

$f(B) = g(B) + h(B) = 6 + 8 = 14$

$f(F) = g(F) + h(F) = 3 + 6 = 9$

Since $f(F) < f(B)$, so it decides to go to node F.

Path : A → F.

Step:2 :- node G & H can be reached from node F

$f(g) = g(g) + h(g) = (3+1) + 5 = 9$ ↵

$f(h) = g(h) + h(h) = (3+7) + 3 = 13$

Since $f(g) < f(h)$. So it decides to go node G

path A → F → G

Step 3 :-

Node I can be reached from node G

$f(I) = g(I) + h(I) = (3+1+3) + 1 = 8$

I is the only path available so the path :-

$A \rightarrow F \rightarrow G \rightarrow I$

So it decide to go to node I.

Step 4 :- Node E, H and node J can be reached from I :-

$f(E) = g(E) + h(E) = (3+1+3+5) + 3 = 15$

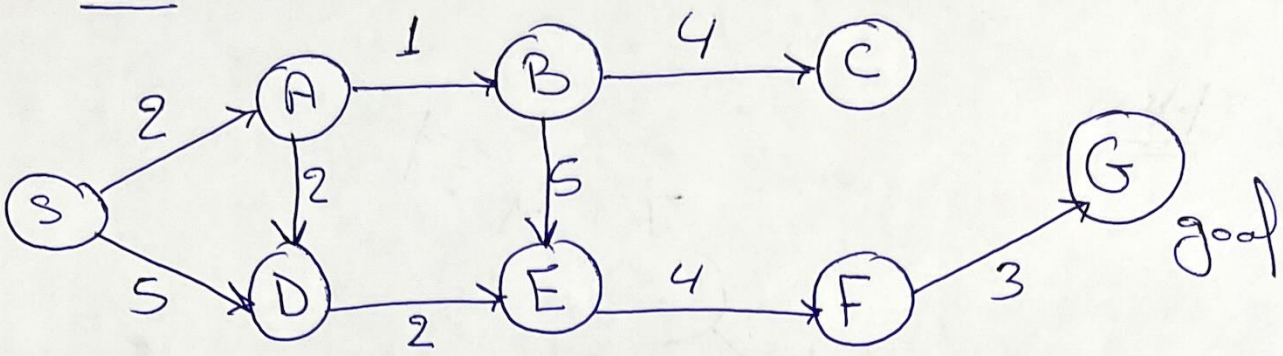
$f(H) = g(h) + h(h) = (3+1+3+2) + 3 = 12$

$f(J) = g(J) + h(J) = (3+1+3+3) + 0 = 10$

Since $f(J)$ is least. So it decides to go to node J :-

Path :- $A \rightarrow F \rightarrow G \rightarrow I \rightarrow J$.

HW :-

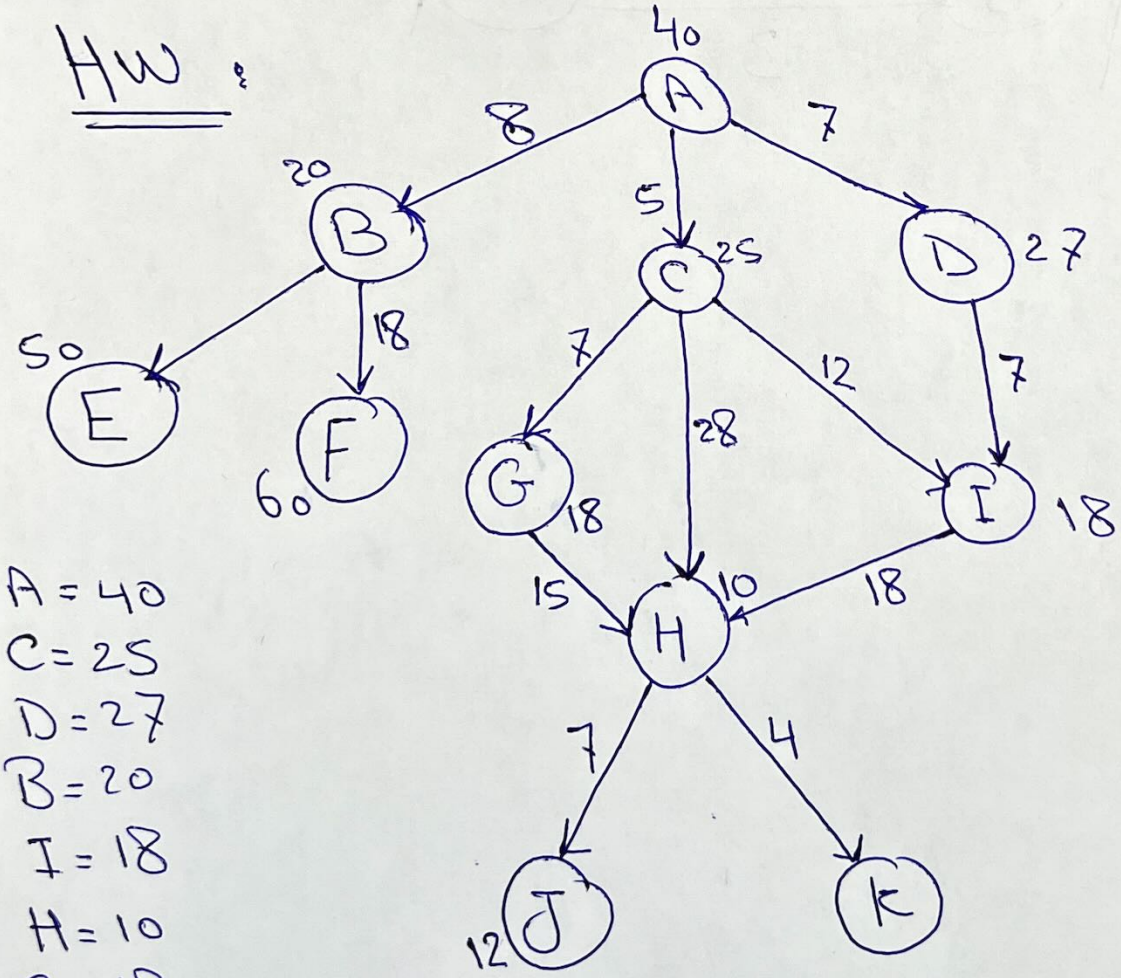


h(n) as following :-

C = 4 B = 6.7 A = 10.4 S = 11 D = 8.9

E = 6.9 F = 3

HW :



A = 40

C = 25

D = 27

B = 20

I = 18

H = 10

G = 18

E = 50

F = 60 , J = 12

HW